

# **Sissejuhatus infotehnoloogiasse**

- Arhitektuur ja perekonnad
- Komponendid: terviksüsteemid, serverid, teegid
- Paradigmad, vaated, rakendusvaldkonnad
- Arendusvahendid
- Tarkvara & tootmine: ajaloolised faasid
- Võrgunduse areng
- Süsteemide integratsioon
- Vabataarkvara

# Tarkvara arhitektuur

---

- Arhitektuuri all mõeldakse IT-s:
  - mingi süsteemi tehnoloogilisi põhimõtteid
  - millisteks suurteks osadeks süsteem jaotub
  - kuidas osad omavahel suhtlevad
  - milliseid suuri valmistükke süsteem kasutab
- Tarkvaras on sellisteks küsimusteks näiteks:
  - Mis opsüsteemi alla rakendus teha
  - Kas hoida infot lihtsalt failides või andmebaasis
  - Millist veebiserverit ja kuidas täpselt kasutada
  - Kas pöörduda andmebaasi poole otse või XML või json vahekihi abil
  - Mis keeles/keeltes rakendus teha
  - Millist kasutajaliidese teeki kasutada
  - ... jne ....

Tarkvara lihtne analoogia ehitusega ei ole eriti pädev:

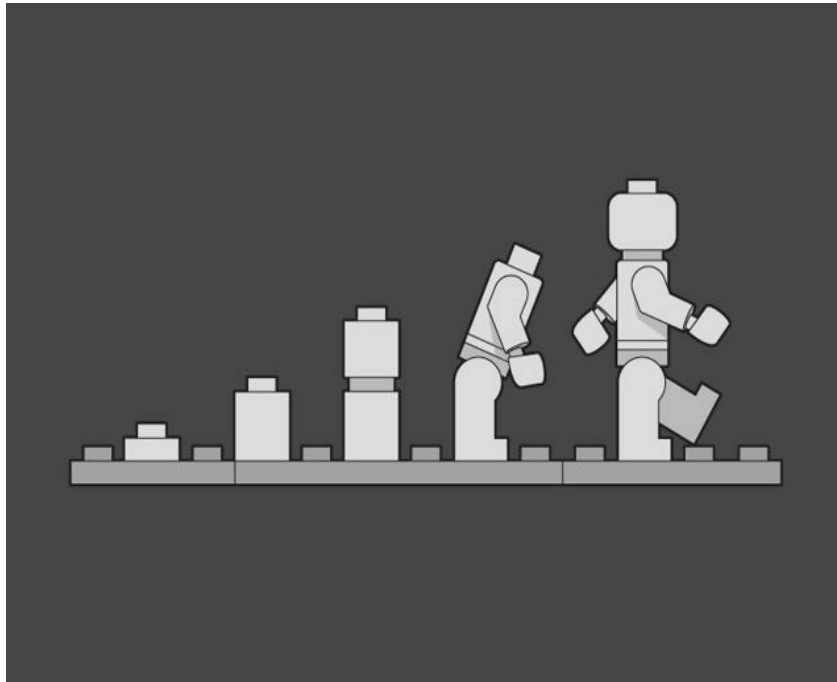
- Tarkvara on kõige keerulisem asi maailmas: tarkvarasüsteemidel on kõige rohkem osi
- Arendaja kirjutatud programm ongi tegelikult ehitusjoonis, mitte ehitus: ehitus (töötav tarkvara) tehakse kompilaatorite, linkurite jms abil programmist automaatselt.
- Programmeerija “analoog” on arhitekt/ehitusinsener.
- Tarkvara arenduses ei ole nõ lihtsaid “ehitusmehi” vaja.
- Täiendava tööjõu lisamine projekti käigus teeb projekti täitmise reeglina veel aeglasemaks.

# Tähelepanekud: II

---

Tarkvara ei panda kokku “nagu lego blokkidest”:

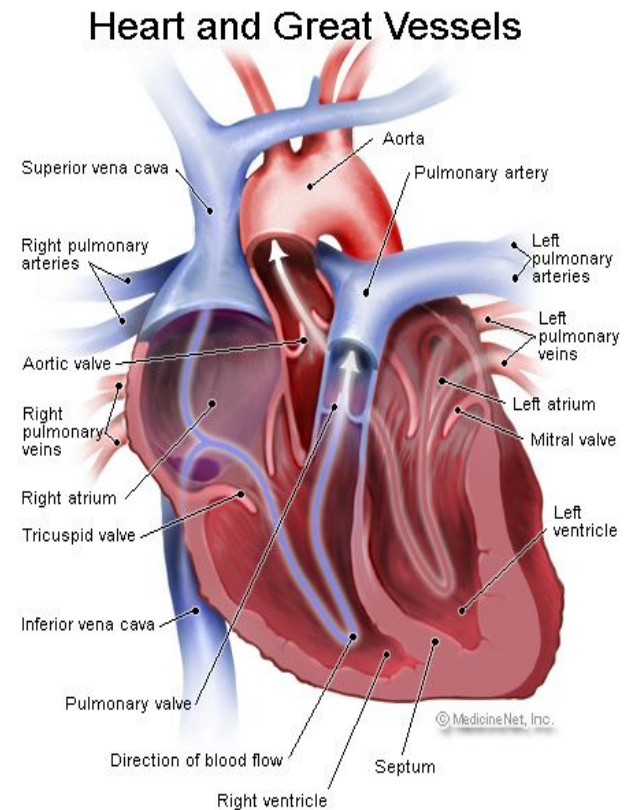
- Lego blokke saab ühendada väga lihtsalt, ja neil ei ole sisemist keerulist ehitust



# Tähelepanekud: II

Tarkvara ei panda kokku “nagu lego blokkidest”:

- “Tarkvarablokid” on väga keerulised süsteemid, ja nende ühendamine tähendab nõ tavapärast programmeerimist



# Tähelepanekud: III

---

Arendamise ja haldamise lihtsus on kõige tähtsam.

- Valiku juures on otsustav see, mida arendajad/haldajad kõige paremini oskavad kasutada.
- Lisatükkide kasutamist ilma selge vajaduseta tuleks vältida.
- Abstraktsioonid tilguvad läbi.
- Ei ole olemas iga juhu jaoks sobilikke tehnoloogiaid/komponente

# Tarkvarasüsteemide komponendid

---

- **Tarkvarasüsteemid** ehitatakse reeglina mitmesuguste komponentide kokkupaneku teel. Neid komponente võib klassifitseerida - näiteks - järgmisel viisil:
  - Terviklikud lõppkasutaja-rakendusprogrammid
  - Suured “valmiskomponendid”
  - Teegid



# Tarkvarasüsteemide komponendid: 1

---

- **Terviklikud lõppkasutaja-rakendusprogrammid**  
(tekstitöötlus, raamatupidamise tarkvara, malemängu programmid jms), mida tihti, kuid mitte alati, saab juhtida ja mõne teise tarkvarapaketiga programmiliselt siduda.

# Tarkvarapakettide komponendid: 2

- **Suured valmiskomponendid** (andmebaasiserverid, www-serverid, mailiserverid, graafikaserverid nagu X11, operatsioonisüsteem ise jne)

Neid programme saab põhimõtteliselt küll kasutada iseseisvalt, ilma millegita sidumata, kuid tüüpiliselt ehitatakse lõppkasutaja jaoks eraldi spetsiaalne rakendus, mis omakorda kasutab neid suuri valmiskomponente, a la:

```
....
cursor=con.cursor()
sqlstr="""select clientid, clientname
           from clientbase
           where clientname like '%Jaan%'"""
cur.execute(sqlstr)
results=cur.fetchall()
for i in results
    print "id: ",i[0],"name: ",i[1]
.....
```

# Tarkvarapakettide komponendid: 2

- **Suured valmiskomponendid** (andmebaasiserverid, www-serverid, mailiserverid, graafikaserverid nagu X11, operatsioonisüsteem ise jne)

Neid programme saab põhimõtteliselt küll kasutada iseseisvalt, ilma millegita sidumata, kuid tüüpiliselt kasutatakse neid loodavate rakenduste komponentidena, a la:

```
.....
cursor=con.cursor()
sqlstr="""select clientid, clientname
           from clientbase
           where clientname like '%Jaan%' """
cur.execute(sqlstr)
results=cur.fetchall()
for i in results
    print "id: ",i[0],"name: ",i[1]
.....
```

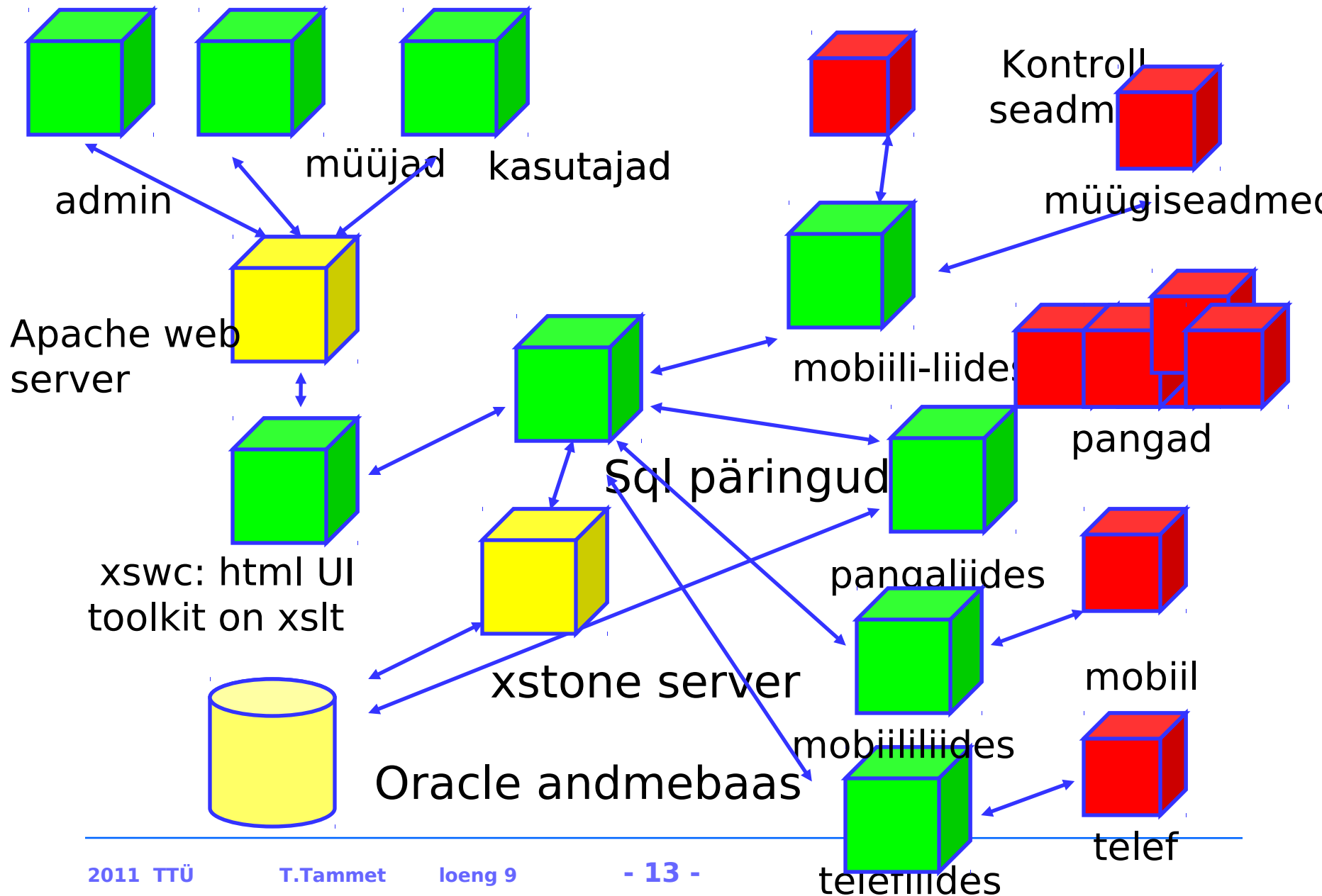
# Tarkvarapakettide komponendid: 3

---

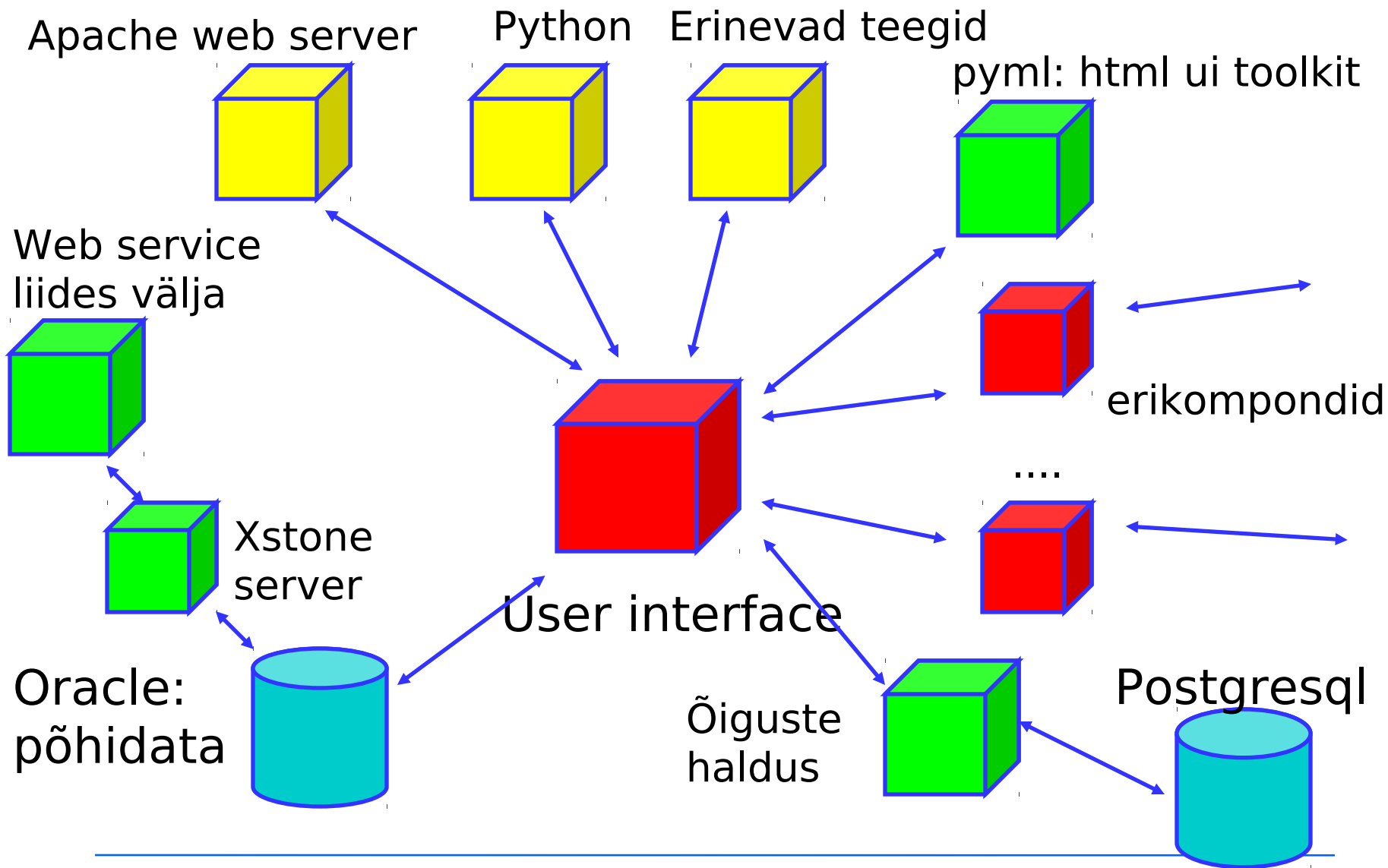
- **Konkreetsed, piiratud funktsioone** (graafika joonistamine ekraanile, aritmeetika, failide lugemine ja kirjutamine, internetiühenduse käivitamine mingile aadressile jms) **realiseerivad väikesed komponendid ja nende komplektid, nn teegid.**

Selliseid komponente levitatakse enamasti komplektis kompilaatorite ja muude tarkvara-arendusvahenditega ning nad on enamasti kasutatavad ainult selle konkreetse programmeerimiskeele ja arendusvahendi koosseisus.

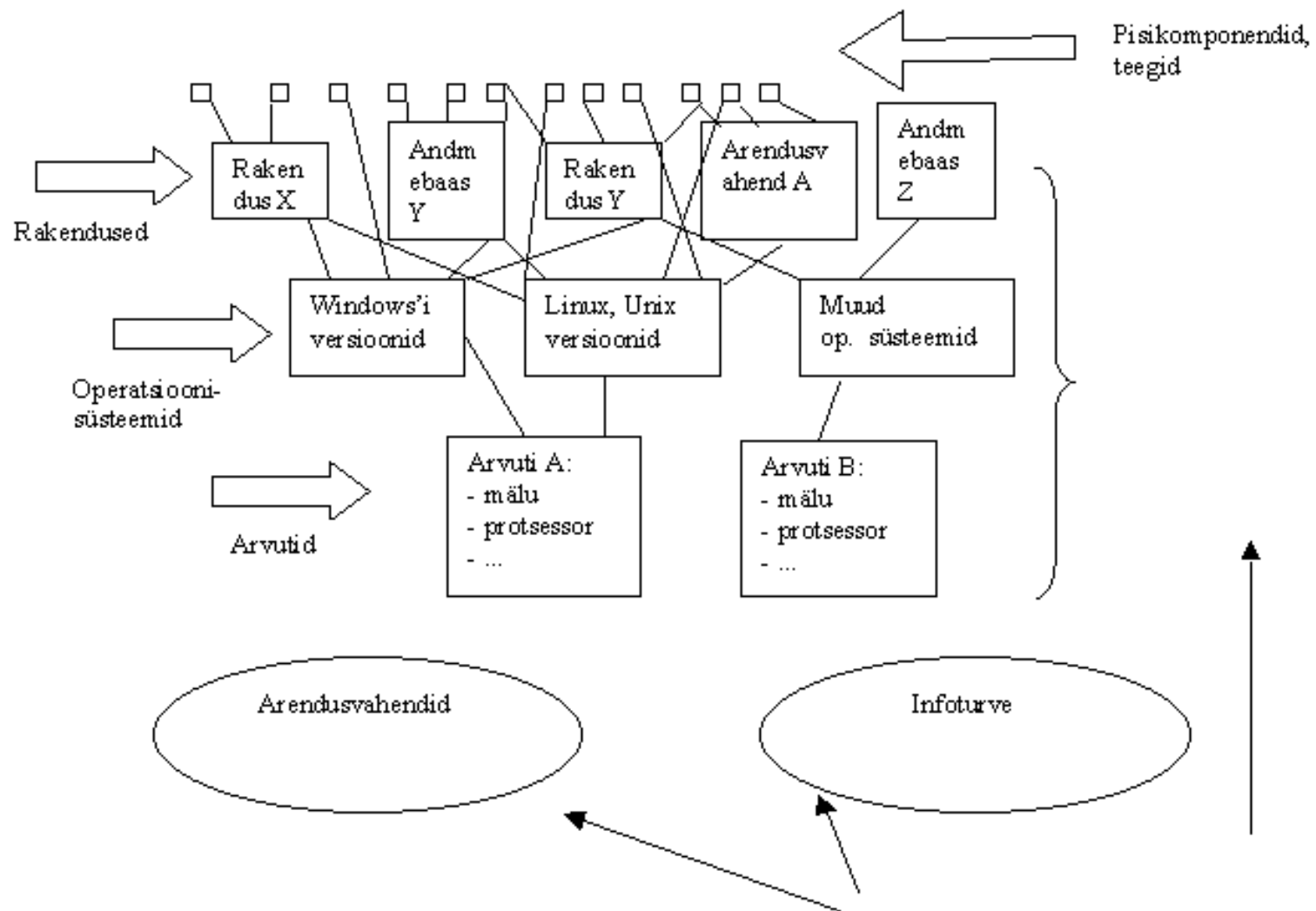
# Arhitektuur: idpileti näide



# Arhitektuur: näide ühest eesti registrist

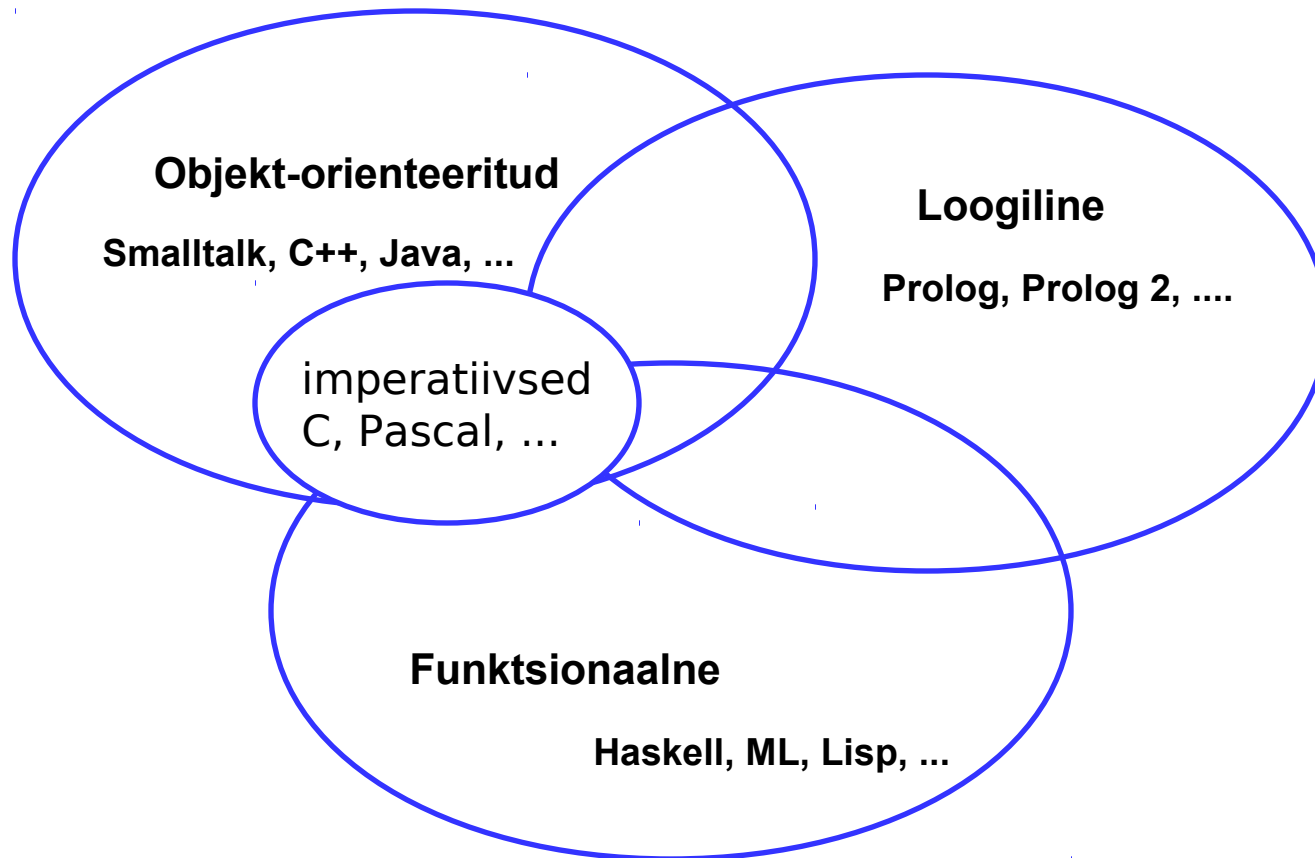


# Süsteemide arhitektuur ja perekonnad



# Programmeerimiskeelte paradigmad

**Vaade 1:** suuremad algoritmikeelte paradigmad  
erivahendite järgi, mida keel pakub:





# Algoritmilised keeled ja kirjelduskeeled

---

## **Vaade 2: algoritmilised ja kirjelduskeeled**

- **Algoritmilised (programeerimiskeeled)**

- C, C++, Basic, COBOL, Java, Lisp, assembler ....

- **Kirjelduskeeled (spetsifitseerimiskeeled)**

- HTML, RDF, XML, SQL, ...

- **Korraga mõlemat: loogika**

- **Praktiline süntees:**

- algoritmilised keeled manipuleerivad kirjelduskeelega abil antavate objektidega, algoritmilise keele abil töödeldakse kirjeldusi
- näited: Javascript ja HTML., C ja SQL, ....

## **Vaade 3: rakendusteegid**

- **Enamik keeli EI SISALDA STANDARDSEID TEEKE või need teegid on äärmiselt piiratud**

Näide: C teek sisaldab stringitöötlust, failitöötlust, trükkimist, veel paari analoogilist kategooriat.

- Reeglina on võimsad rakendusteegid (graafika, aknad, hiir, võrguühendused, paralleelprotsessid jne ...) ebastandardised, piiratud teatud riistvara ja opsüsteemiga (Windows, Linux, iOS, Android, ...)
- Pea ainus erand: Java

# Praktilised rakendusvaldkonnad

---

- Universaalne, välja arvatud tippkiirust nõudvad või operatsioonisüsteemi-rakendused: Java
- Maksimaalset kiirust nõudvad rakendused, süsteemprogrammeerimine: C, C++
- Windowsi rakenduste programmeerimine: C#, VisualBasic, C, Java, ...
- Brauseri programmeerimine: Javascript, (Flash, Java)
- Serveri programmeerimine: PHP, Python, Java, Perl, Ruby, C, ..
- Spetsiifilised rakendused: vastavalt vajadusele

# Praktilised rakendusvaldkonnad

---

- Universaalne, välja arvatud tippkiirust nõudvad või operatsioonisüsteemi-rakendused: Java
- Maksimaalset kiirust nõudvad rakendused, süsteemprogrammeerimine: C, C++
- Windowsi rakenduste programmeerimine: C#, VisualBasic, C, Java, ...
- Brauseri programmeerimine: Javascript, (Flash, Java)
- Serveri programmeerimine: PHP, Python, Java, Perl, Ruby, C, ..
- Spetsiifilised rakendused: vastavalt vajadusele

# Praktilised rakendusvaldkonnad

---

- Universaalne, välja arvatud tippkiirust nõudvad või operatsioonisüsteemi-rakendused: Java
- Maksimaalset kiirust nõudvad rakendused, süsteemprogrammeerimine: C, C++
- Windowsi rakenduste programmeerimine: C#, VisualBasic, C, Java, ...
- Brauseri programmeerimine: Javascript, (Flash, Java)
- Serveri programmeerimine: PHP, Python, Java, Perl, Ruby, C, ..
- Spetsiifilised rakendused: vastavalt vajadusele

Peamine:

- Kompilaator
- Interpretaator

Programmi kirjutamiseks:

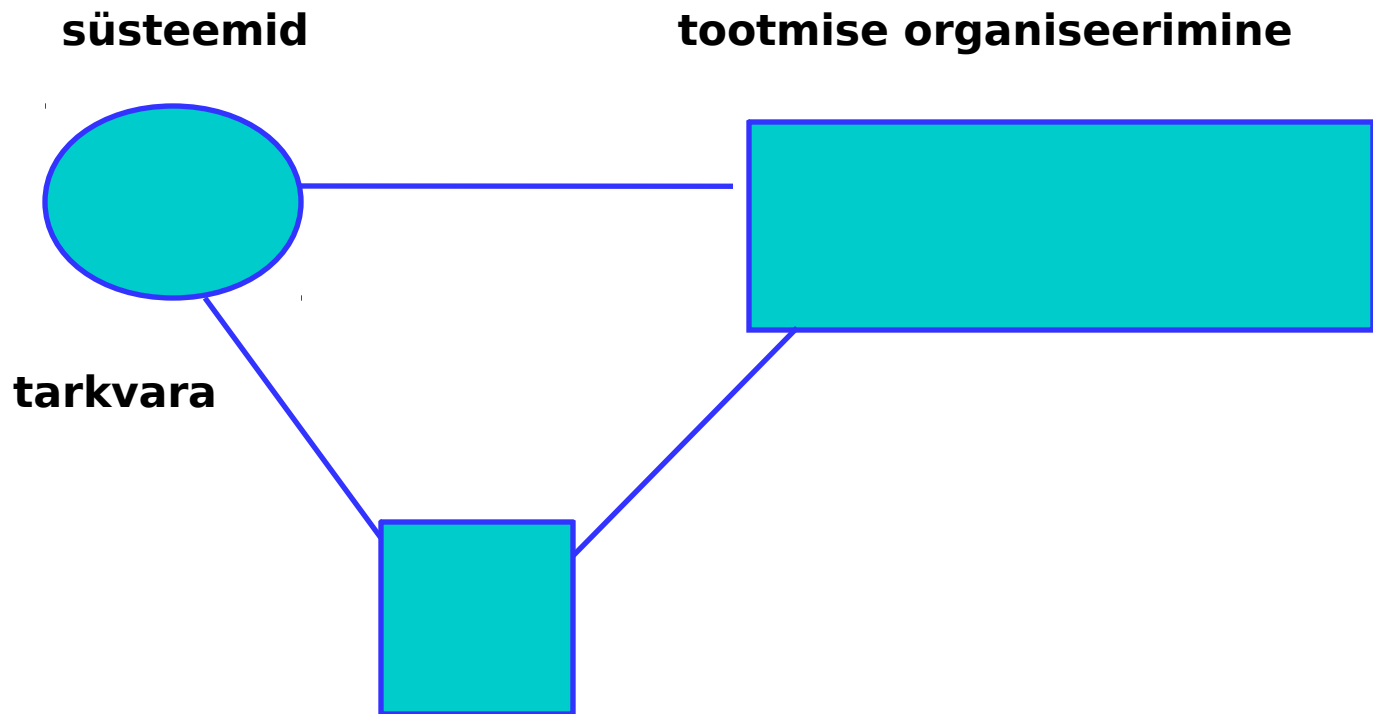
- Sobivad tekstiredaktorid
- Visuaalsed arendusvahendid

Suure hulga lähtekoodi haldus:

- Versioonikontroll (CVS, subversion, git jne)
- Kompileerimissüsteemid (make, automake, Ant, ...)

# IT süsteemide paradigmade areng

**Ajaloolised vaated:**



# arhitektuurifaasid

---

**1945-1970**

**1970-1995**

**1995-2020?**

## **A) Süsteemide arhitektuurifaasid**

Suurarvutid - Mikroarvutid - Võrgusüsteemid

## **B) Tarkvaraplatvormide arhitektuurifaasid:**

assembler, puhtad keeled -  
teegid, arendusvahendid, komponendid -  
komponentide sidumine

## **C) Tootmise organiseerimise arhitektuurifaasid:**

suurfirma, avatud - väikefirma, suletud -  
vabad komponendid, sidumine, hooldus



# Priorities for software development

---

Three main consumers of time and effort:

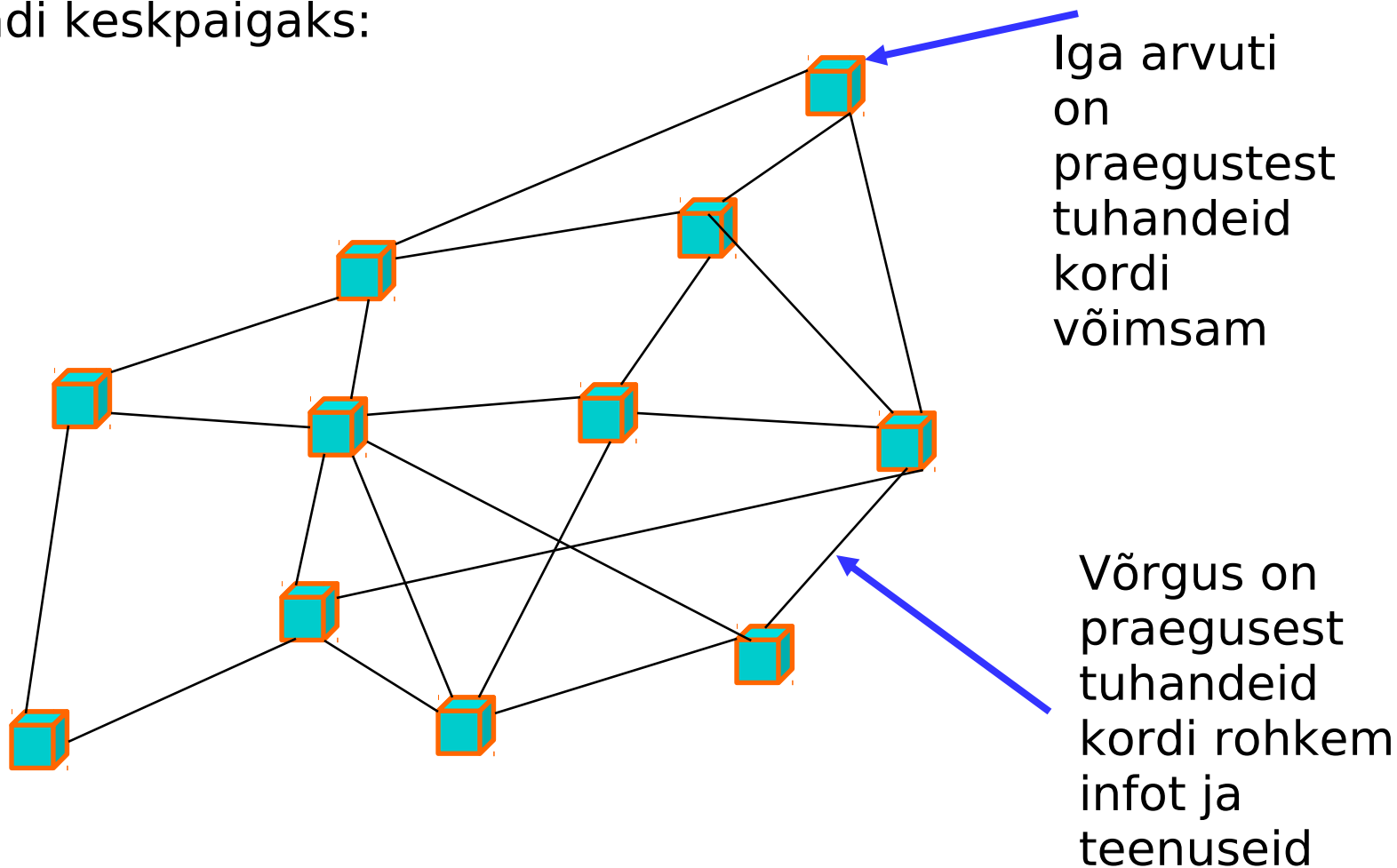
- Understanding the business processes and needs.
- Understanding the exact contents of existing data.
- Writing code.

The second component - understanding existing data - is growing and will keep growing for foreseeable future.

Why?

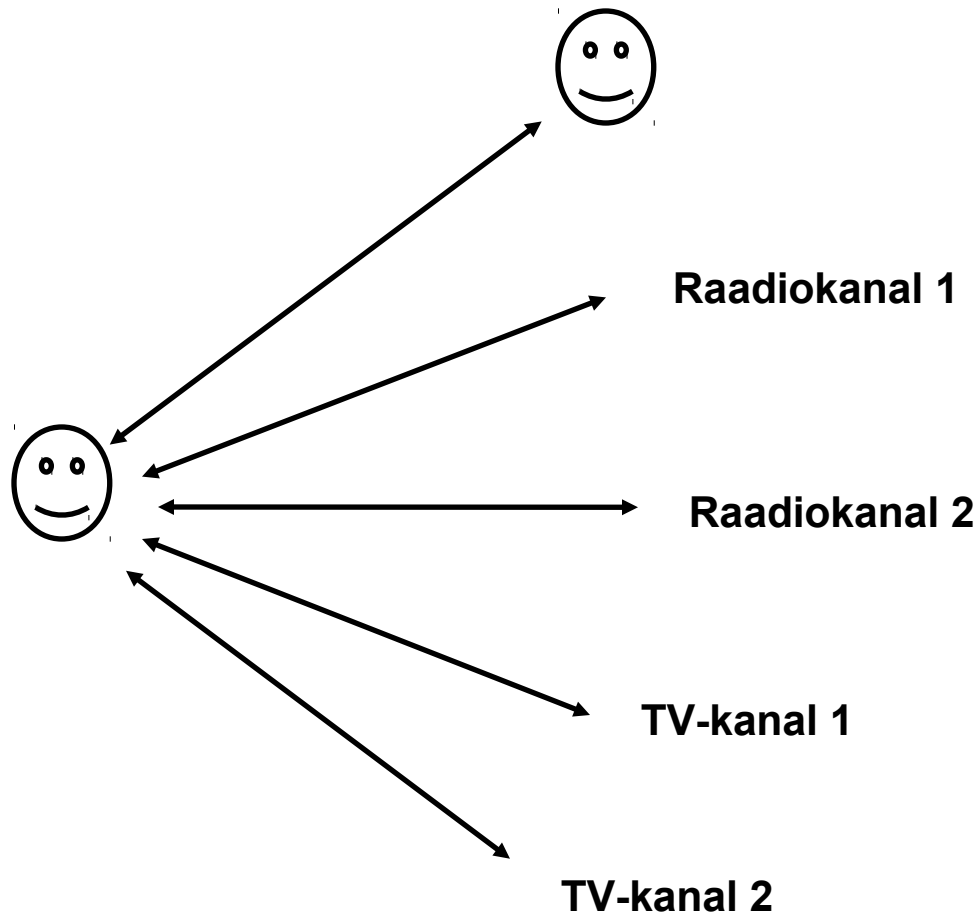
# Intelligentne võrk!?

Sajandi keskpaigaks:

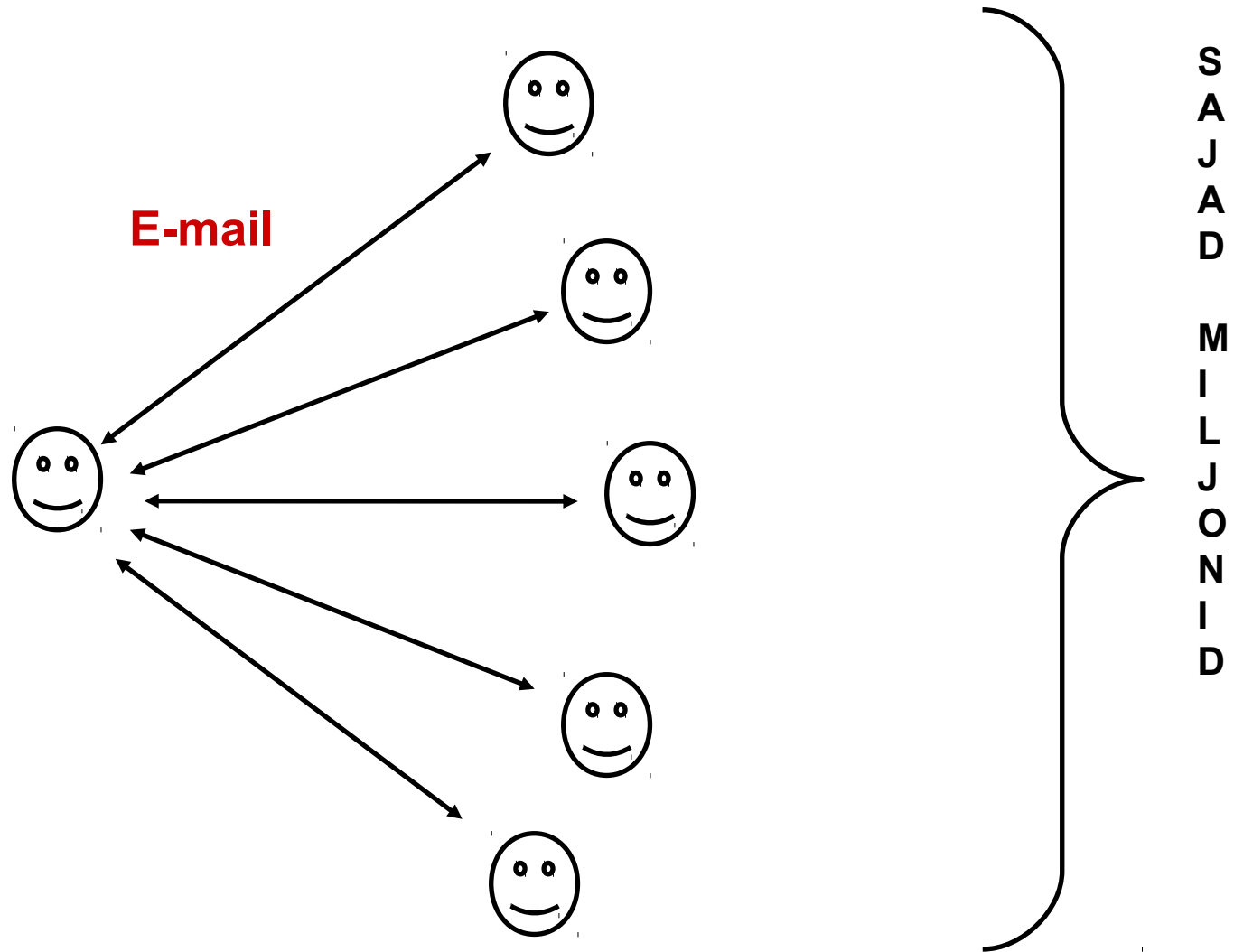


# Internet 0: inimene <--> veidi inimesi, raadio, TV

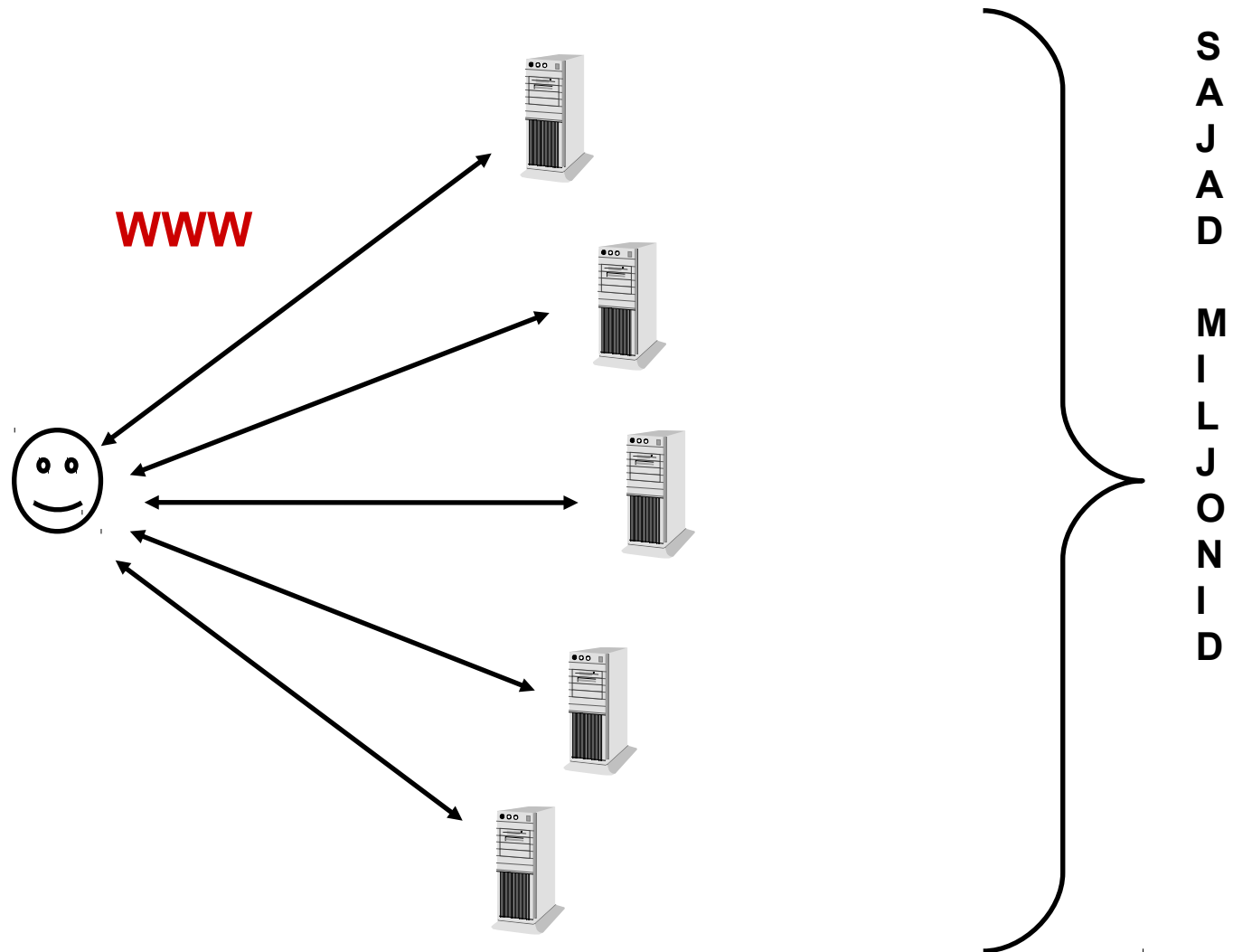
---



# Internet 1: inimene <--> hulga inimesi

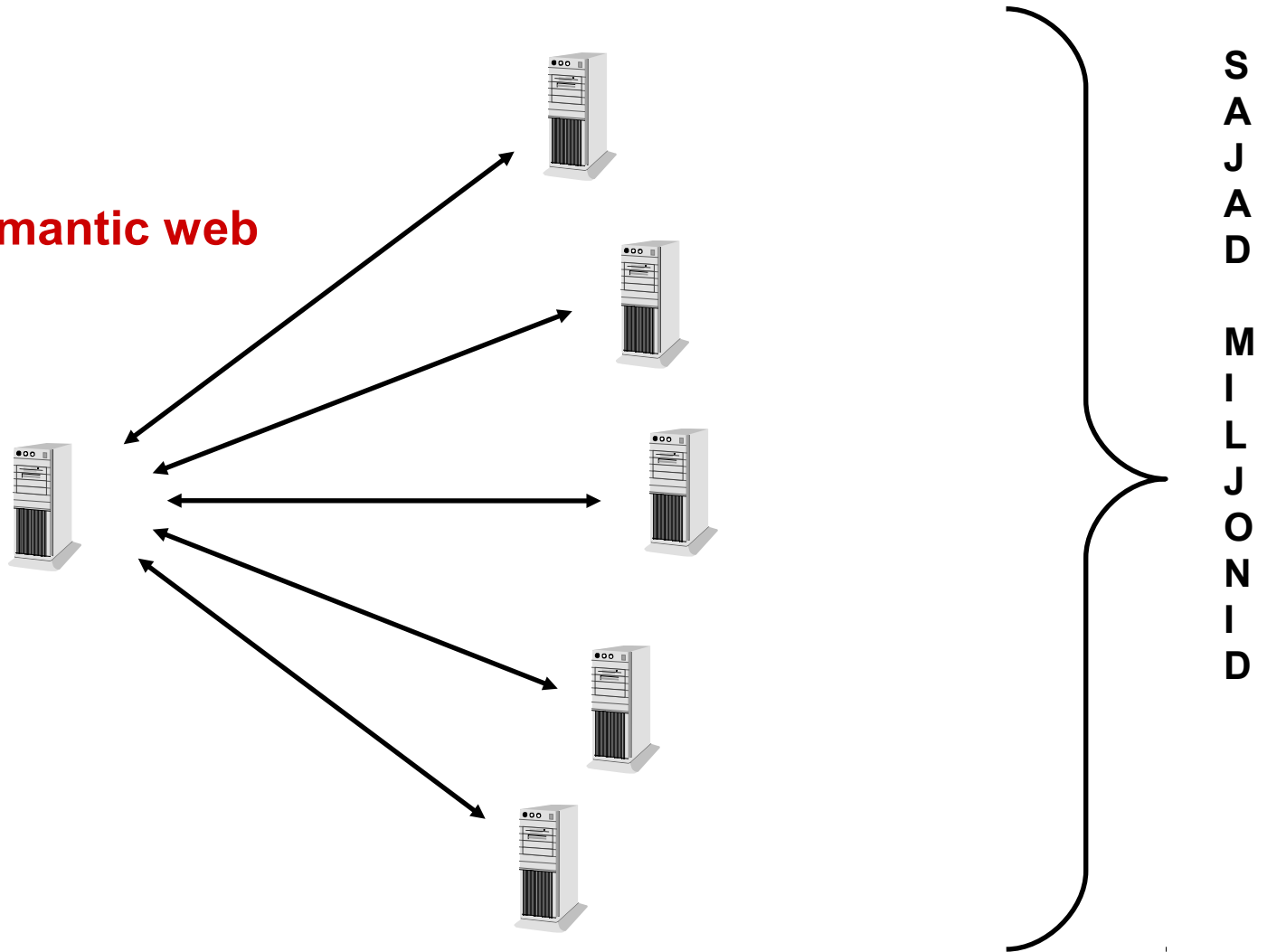


# Internet 2: inimene <--> hulga masinaid

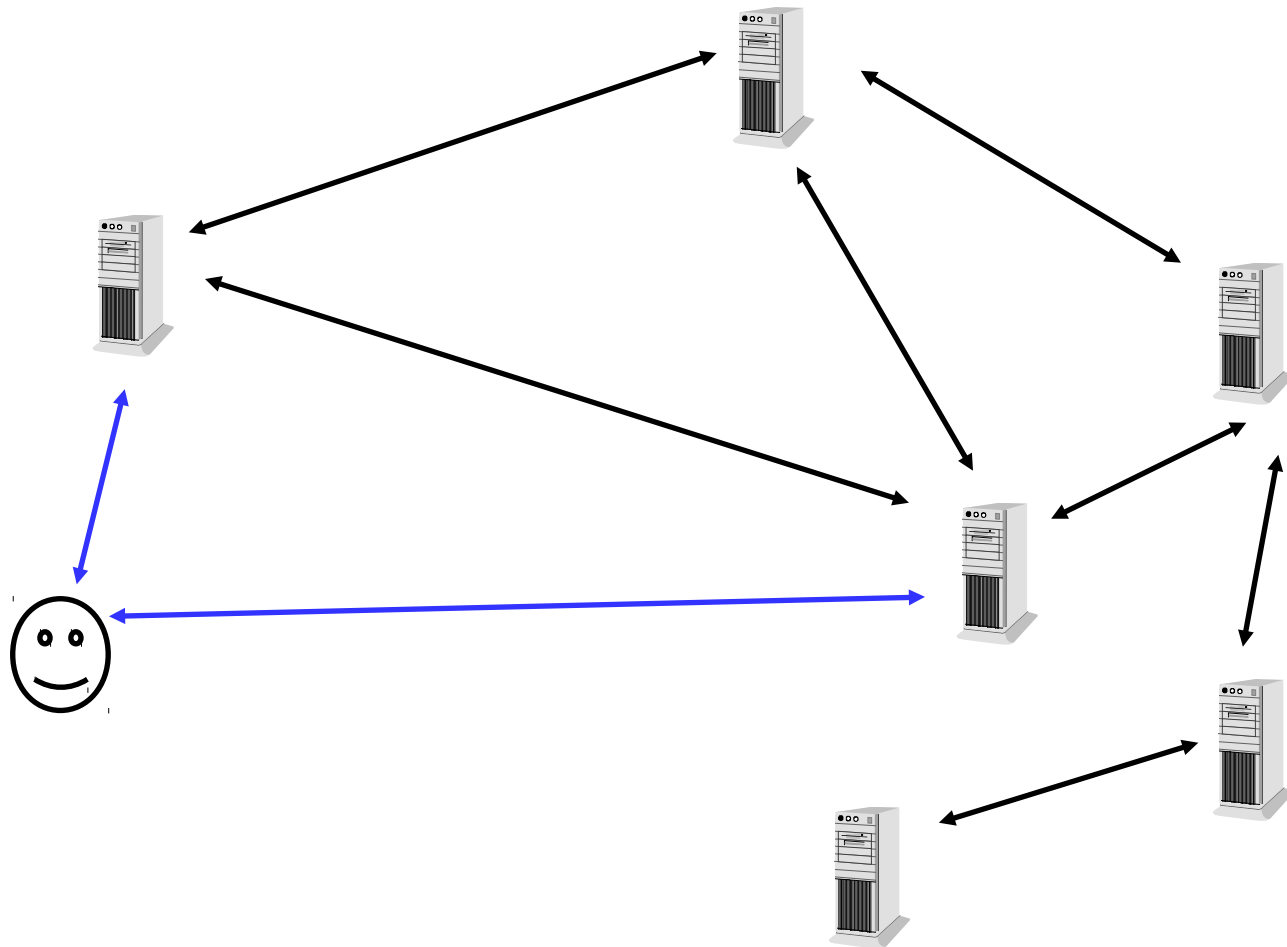


# Internet 3: masinad <--> hulga masinaid

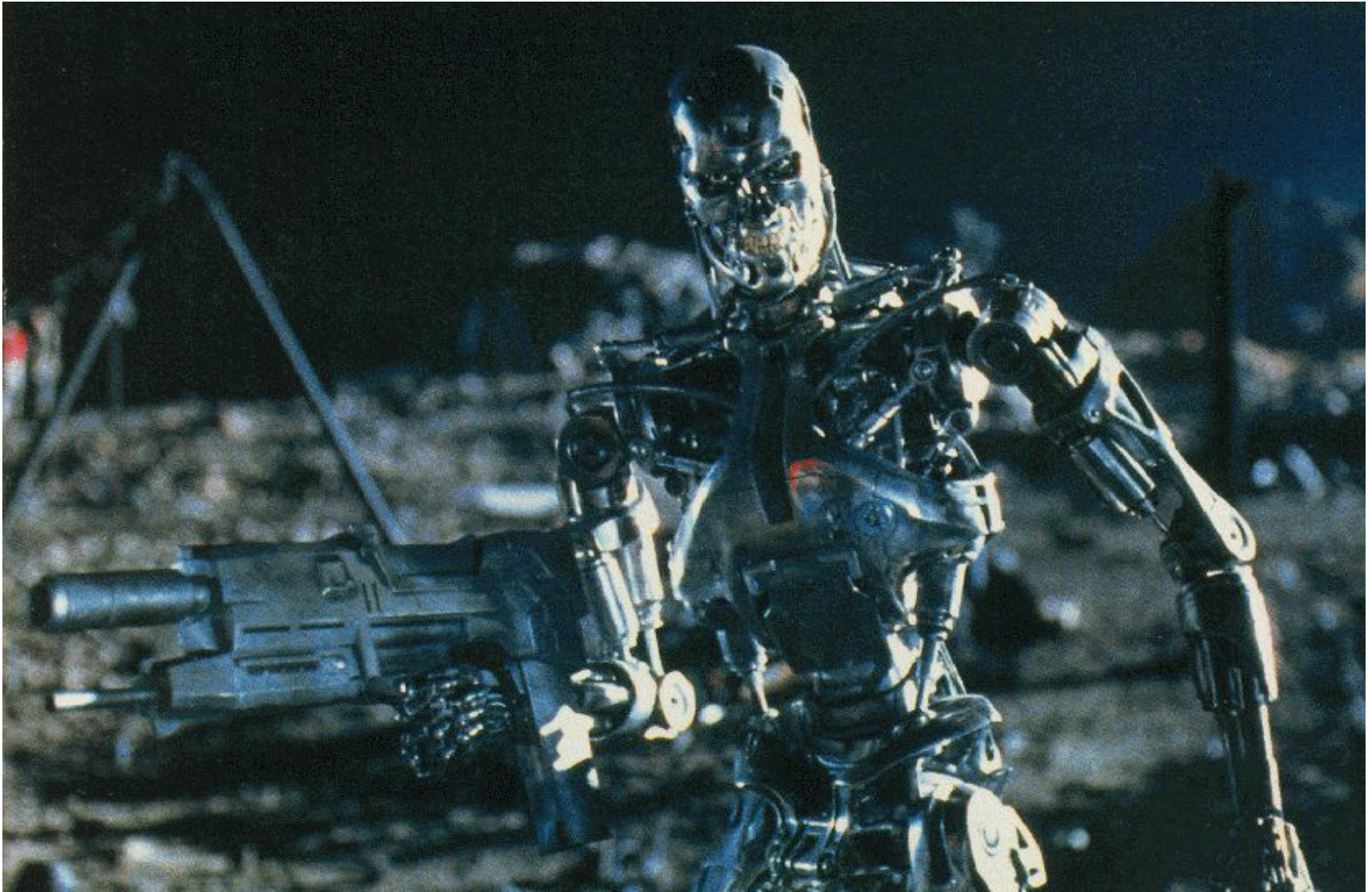
**XML + Semantic web**



# Internet 3: inimene ??



## Internet 4: Skynet?

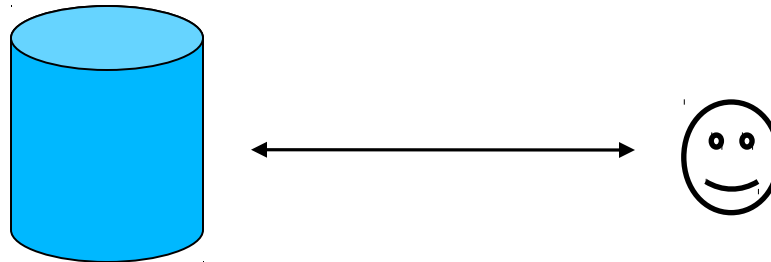




# Architecture I

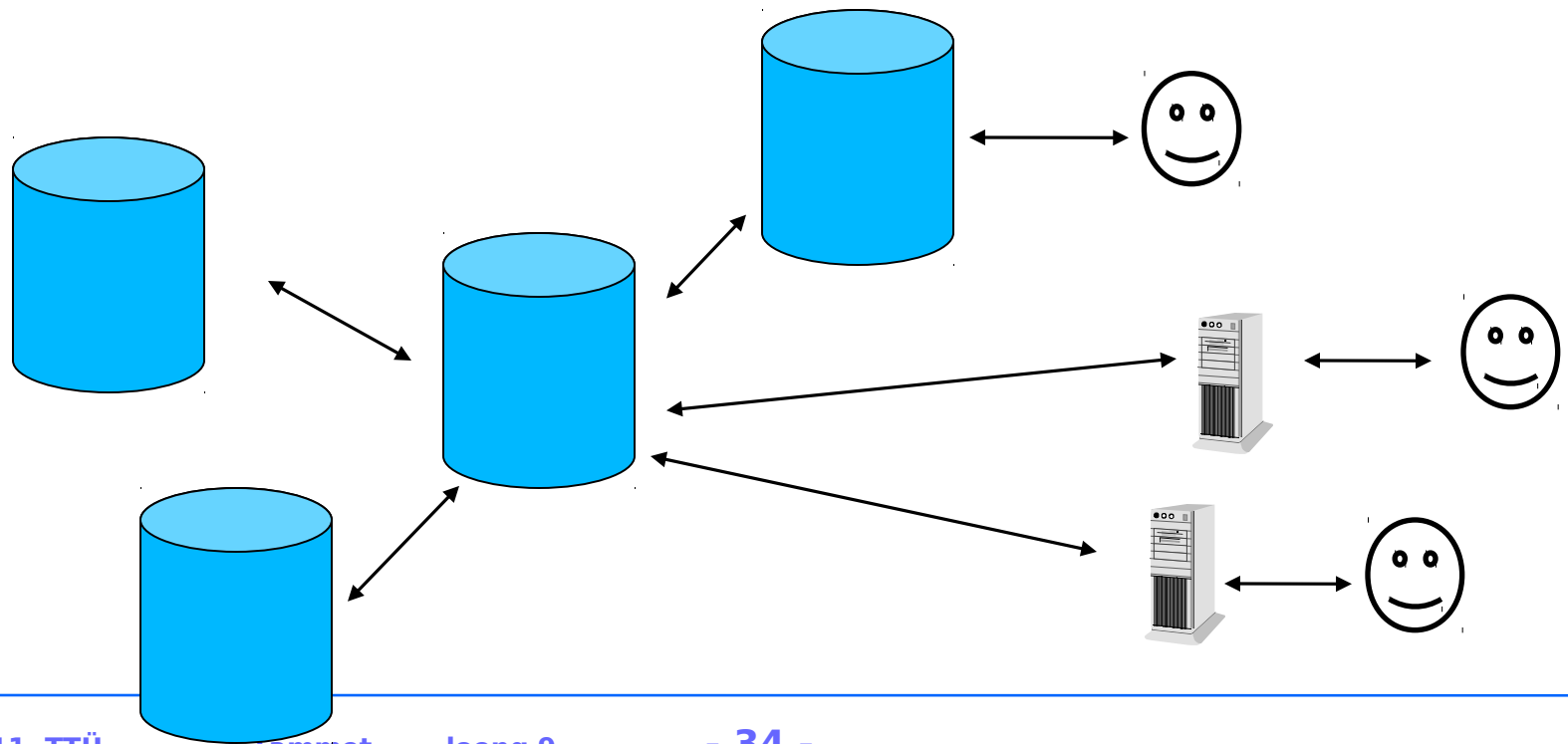
---

Compact standalone software



# Architecture II

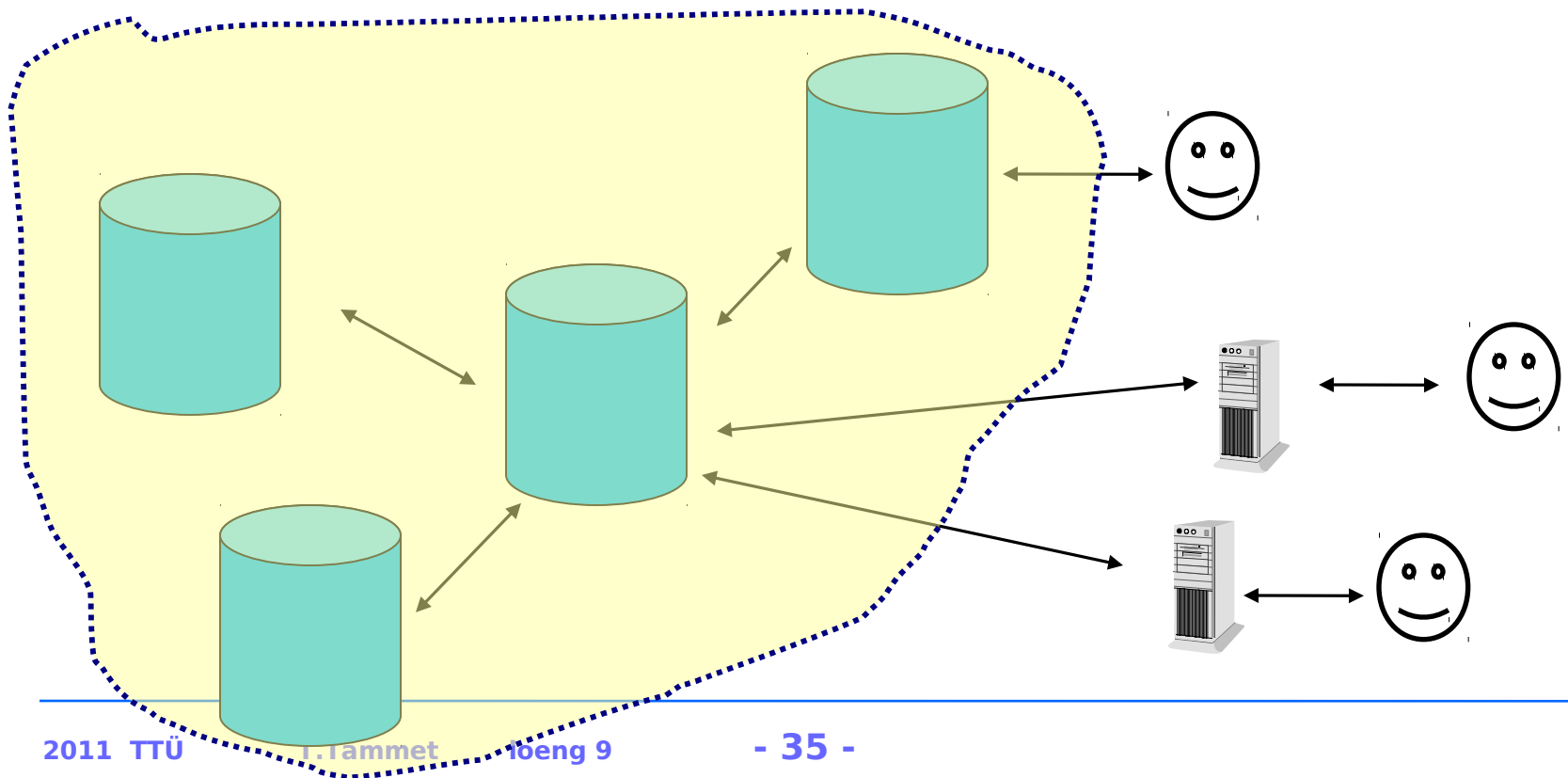
## Classical client-server applications



# Architecture II : similarities to I

All applications controlled by the developer:

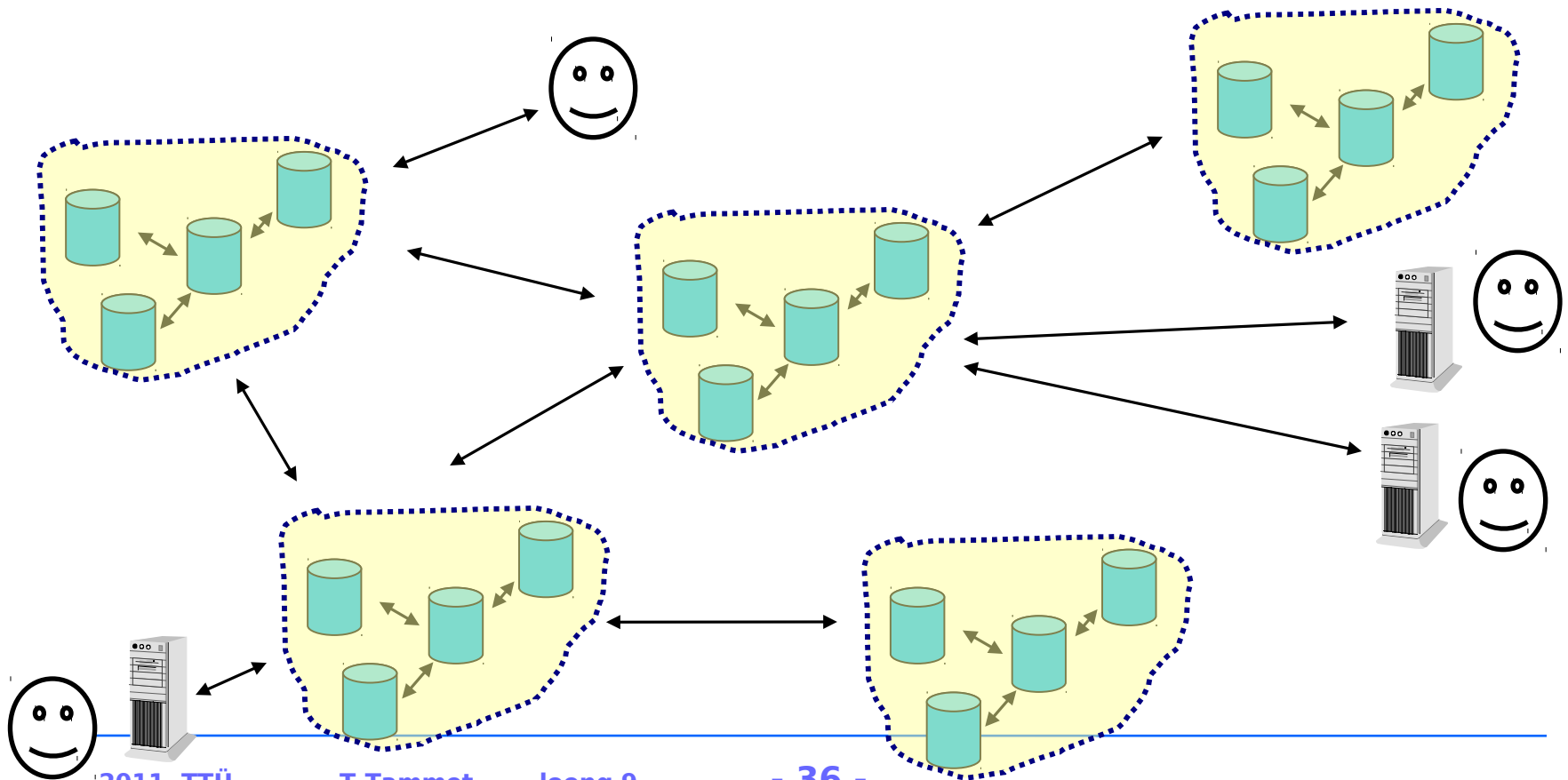
- located in one machine or
- distributed in the machines of the company/client



# Architecture III

Applications **not** controlled by the developer:

- you only have access to them
- system functions like a society of applications



# Architecture III: some examples

---

- Apps based on web services
- Global comparison/search systems:
  - orbitz
  - hinnavaatlus
  - ...
- Estonian public sector: xtee complex queries
- Document management between different organisations
- Large financial apps: banks etc
- Sales systems
- Military apps
- ...

# Architecture III: kill chain example

---

- Context: Yugoslavia, Iraq, Afganistan ...
- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we
  - identify targets,
  - find them,
  - fix it,
  - track it,
  - target
  - execute.
  - check results

# Architecture III: kill chain example


---

- Context: Yugoslavia, Iraq, Afganistan ...
  - Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we
    - identify targets,
    - find them,
    - fix it,
    - track it,
    - target
    - execute.
    - check results
- Every link is a separate country with its own information systems

# Architecture III: kill chain example

---

- Context: Yugoslavia, Iraq, Afganistan ...
- Warfighting integration. We're basically tasked to close the seams in the kill chain. The kill chain is a euphemism for the process by which we
  - identify targets,
  - find them,
  - fix it,
  - track it,
  - target
  - execute.
  - check results



Every link is a separate country with its own information systems  
Countries in the chain are often exchanged or replaced!!



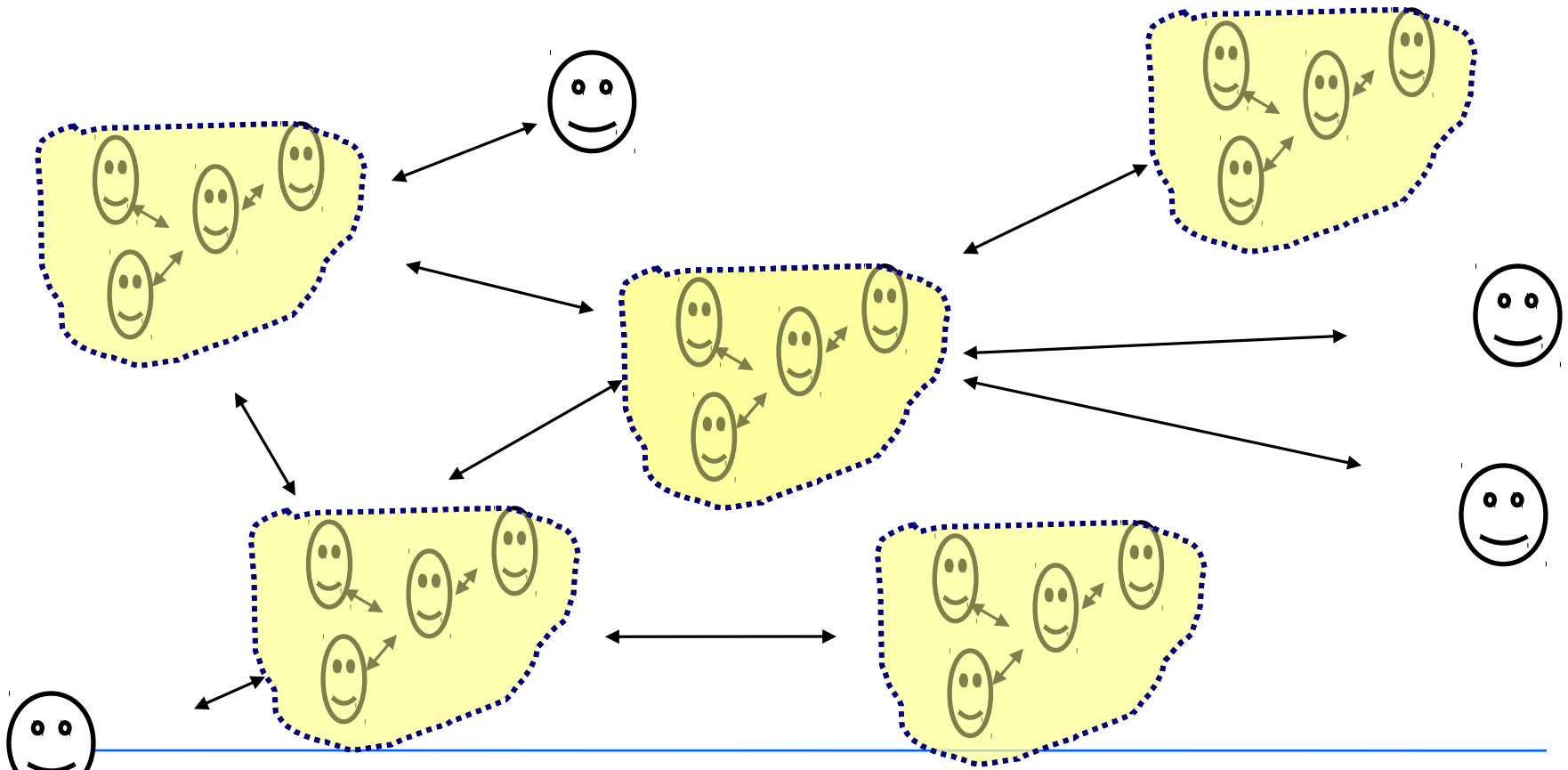
# Main problems for the developer

---

- **Get access** to data from the foreign system
- **Understand** what does this data exactly mean
- **Convert** foreign data to data structures of our own app and vice versa

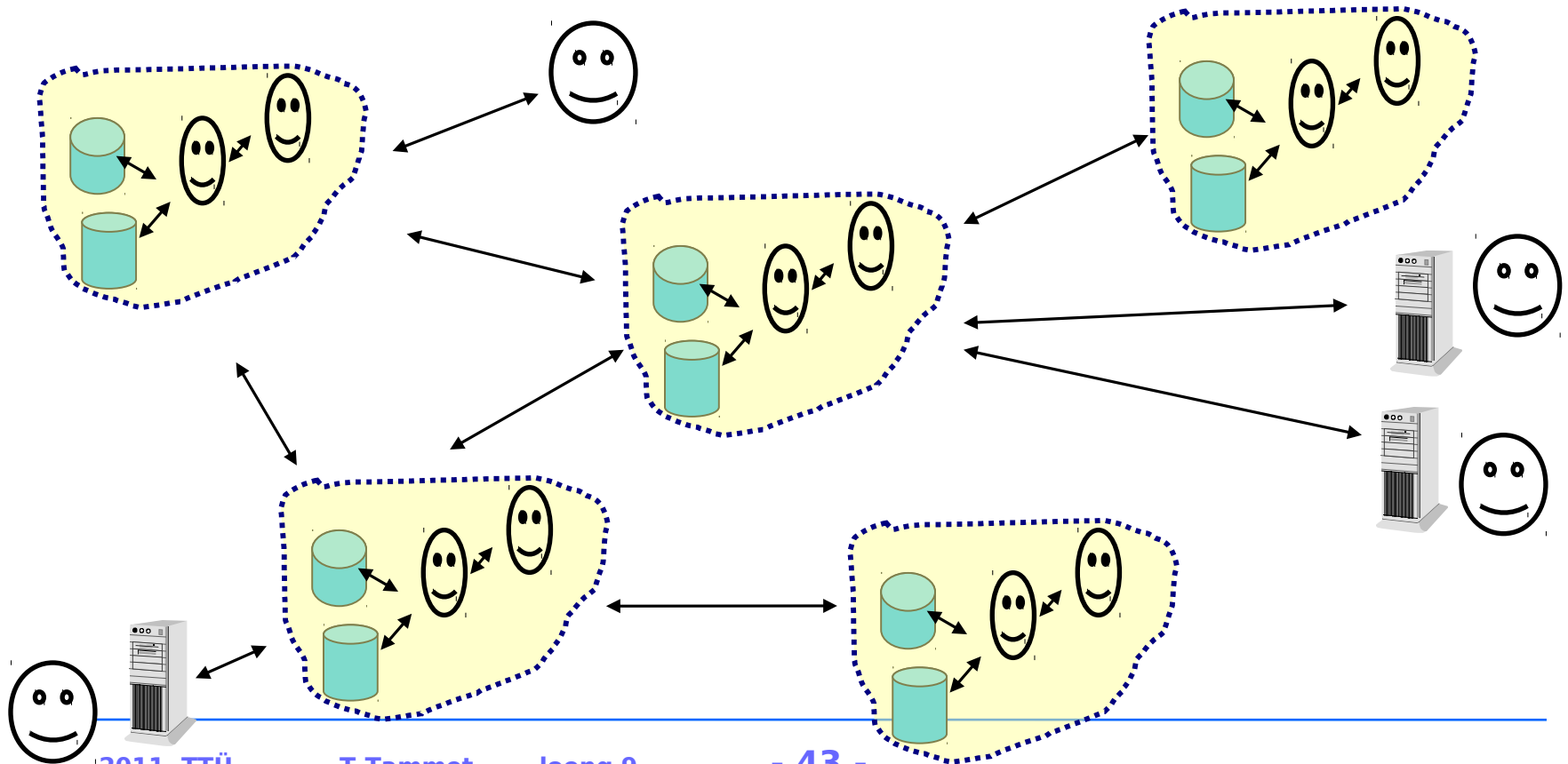
# Getting access and understanding?

Social networking. Connected groups of people:



# Architecture III revisited

Social groups consist of both people and apps



# Philosophy. Society and people.

---

- We do have AI already: society is a large animal with an intelligence of its own.
- **People - society** is just like **cells - animal**
- Society AI benefits from:
  - clever organisation of its brain cells (people)
  - good communication of its brain cells (people)
- Society does not really need very bright people.
- It needs submissive and communicative people.

# Philosophy. Notes.

---

- Both apparently - almost - true:
  - All technologies are turned into guns.
  - All technologies are turned into comm devices.

- **Mis on vabatarkvara:**

- tasuta?
- lahtise koodiga?
- edasimüügiõigusega?
- copyright?

- **Ajalugu, eesmärgid, perspektiivid**

- mis on olnud?
- mis on tulemas?
- äriidee?
- miks vabavara tehakse?
- kas ja millal on vabavara mõistlik kasutada?
- mida vabavara on endaga kaasa toomas?

## GNU ideoloogia:

- vabadus: primaarne on tarkvara **vabadus**, sekundaarne **tasuta kättesaadavus**
- ausus: ausam on kasutada vabavara kui piraatkopeerida
- teadmiste vabadus: teadmised, tarkvara tahab olla vaba, on loomu poolest vaba - teadmiste ja tarkvara kopeerimine laiendab ühiskonna majanduslikku võimsust, kaotajaid (rumalamaks jääjaid) pole
- raha saab teenida ka tarkvara
  - toetades ja installeerides ja levitades
  - tellitud täiustusi ja modifikatsioone ehitades
  - ehitades sellist vabavara, mida klient soovib
- motivatsioon senistes kogustes tarkvara luua väheneb?
  - vastuargument: programmeerimine on **põnev**
  - vastuargument: ehk polegi meil nii palju kallilt makstud programmiste vaja?
  - vastuargument: vabanenud programmeerijate kaader saab teha uusi, senisest keerulisemat softi, mille jaoks seni aega ega ressursi ei jätkunud

- **Vaatame eraldi GPL ja LGPL litsentse**